In the Claims:

Please amend claims 10 and 17, and cancel claim 19, all as shown below.

1 – 9. (Cancelled)

10. (Currently Amended): A method for processing an invocation using a dynamically generated wrapper, comprising:

receiving an invocation by a wrapper object, the wrapper object instantiated from a wrapper class, the wrapper class extended from a superclass which implements a ~~predefined~~ wrapper interface that includes a pre-invocation handler and a post-invocation handler, the invocation directed to a wrapped resource adapter by an application;

initiating pre-processing by the wrapper object, wherein the pre-processing code includes calling the pre-invocation handler, wherein the pre-invocation handler is configured to execute server-side code, wherein the server-side code includes transaction processing code;

calling the wrapped resource adapter by the wrapper object;

receiving a result from the wrapped resource adapter by the wrapper object;

initiating post-processing by the wrapper object, wherein post-processing including calling the post-invocation handler, wherein the post-invocation handler is configured to perform post-processing server side tasks, wherein the post-processing server-side tasks include transaction management; and

provide the result to the application.

11. – 16. (Canceled)

17. (Currently Amended): A method for dynamically generating a wrapper object, comprising:

receiving a resource adapter class at an application server;

performing reflection on the resource adapter class to identify interfaces implemented by the resource adapter class;

dynamically generating a wrapper class at runtime that extends from a superclass, wherein the superclass implements a ~~predefined~~ wrapper interface that includes a pre-invocation handler and a post-invocation handler, and the wrapper class implements the interfaces identified through reflection;

instantiating a wrapper object from the wrapper class;

initiating pre-processing by the wrapper object, wherein the pre-processing code includes calling a pre-invocation handler, wherein the pre-invocation handler is configured to execute server-side code, wherein the server-side code includes transaction processing code; and

providing the wrapper object to an application that requires support for the interfaces implemented by the resource adapter class.

18 - 19. (Canceled)

20. (Previously Presented): The method of claim 17 further comprising:

initiating post-processing by the wrapper object, wherein post-processing including calling a post-invocation handler, wherein the post-invocation handler is configured to perform post-processing server side tasks, wherein the post-processing server-side tasks include transaction management.

21. (Previously Presented): The method of claim 10, wherein the wrapper object is a proxy generated at runtime and acts as a delegate for an underlying vendor object.

22. (Previously Presented): The method of claim 10, wherein the wrapper object is used to intercept method invocations from an application program to a vendor object and provide for execution of server side tasks in a pre-invocation handler and a post-invocation handler.

23. (Previously Presented): The method of claim 10, wherein the wrapper object is used to intercept a method invocation against the vendor object.

24. (Previously Presented): The method of claim 10, wherein the wrapper object provides for server side tasks to be performed before sending a wrapped result to the application.

25. (Previously Presented): The method of claim 10, wherein the wrapper object is dynamically generated at runtime by a wrapper factory on an application server.

26. (Previously Presented): The method of claim 10, wherein retrieved meta information from performing reflection allows an application server to dynamically generate a wrapper class that perfectly matches the vendor class.

27. (Previously Presented): The method of claim 10, wherein a wrapper class includes all public interfaces implemented by a vendor class and required by the application.

28. (Previously Presented): The method of claim 10, wherein the application can cast the wrapper object to a vendor interface to access vendor extension methods.

29. (Previously Presented): The method of claim 10, wherein the application server has code for dynamically generating the wrapper.

30. (Previously Presented): The method of claim 10, wherein a wrapper factory uses a static method to dynamically generate a wrapper.

31. (Previously Presented): The method of claim 10, wherein the superclass has a member variable to hold a vendor object, a non-argument constructor to instantiate the wrapper object, and an init method to initialize the wrapper object.

32. (Previously Presented): The method of claim 17, wherein the wrapper object is a proxy generated at runtime and acts as a delegate for an underlying vendor object.

33. (Previously Presented): The method of claim 17, wherein the wrapper object is used to intercept method invocations from an application to a vendor object and provide for execution of server side tasks in a pre-invocation handler and a post-invocation handler.

34. (Previously Presented): The method of claim 17, wherein the wrapper object is used to intercept a method invocation against a vendor object.

35. (Previously Presented): The method of claim 17, wherein the wrapper object provides for server side tasks to be performed before sending a wrapped result to the application.

36. (Previously Presented): The method of claim 17, wherein the wrapper object is dynamically generated at runtime by a wrapper factory on the application server.

37. (Previously Presented): The method of claim 17, wherein retrieved meta information from performing reflection allows the application server to dynamically generate a wrapper class that perfectly matches a vendor class.

38. (Previously Presented): The method of claim 17, wherein the wrapper class includes all public interfaces implemented by a vendor class and required by the application.

39. (Previously Presented): The method of claim 17, wherein the application can cast the wrapper object to the vendor interface to access vendor extension methods.

40. (Previously Presented): The method of claim 17, wherein the application server has code for dynamically generating the wrapper.

41. (Previously Presented): The method of claim 17, wherein a wrapper factory uses a static method to dynamically generate a wrapper.

42. (Previously Presented): The method of claim 17, wherein the superclass has a member variable to hold a vendor object, a non-argument constructor to instantiate the wrapper object, and an init method to initialize the wrapper object.